Secure Packager and Encoder
Key Exchange API Specification
Partner and Customer Guide
Secure Packager and Encoder Key Exchange API Specification: Partner and Customer Guide

Copyright © 2020 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.
# Table of Contents

What Is Secure Packager and Encoder Key Exchange? ................................................................. 1  
   General Architecture ........................................................................................................ 1  
   AWS Cloud-Based Architecture .................................................................................... 1  
   How to Get Started ........................................................................................................ 2  
Are You New to SPEKE? ........................................................................................................... 3  
   Related Services and Specifications ............................................................................... 3  
   Terminology ..................................................................................................................... 3  
Customer Onboarding ............................................................................................................ 5  
   Get On Board with a DRM Platform Provider .............................................................. 5  
   SPEKE Support in AWS Services and Products ............................................................ 5  
SPEKE API Specification ....................................................................................................... 8  
   Authentication ............................................................................................................... 8  
      Authentication for AWS Cloud Implementations ....................................................... 8  
      Authentication for On-premises Products ................................................................ 9  
SPEKE API ........................................................................................................................ 9  
   Customizations and Constraints ................................................................................. 10  
   Standard Payload Components .................................................................................... 11  
   Live Workflow Method Call Examples ....................................................................... 12  
   VOD Workflow Method Call Examples ...................................................................... 16  
   Content Key Encryption ............................................................................................... 18  
   Heartbeat ...................................................................................................................... 21  
   Overriding the Key Identifier (KID) ............................................................................ 21  
Document History .................................................................................................................. 23  
AWS glossary ...................................................................................................................... 24
What Is Secure Packager and Encoder Key Exchange?

Secure Packager and Encoder Key Exchange (SPEKE) defines the standard for communication between encryptors and packagers of media content and digital rights management (DRM) key providers. The specification accommodates encryptors running on-premises and in the AWS Cloud.

Topics
- General Architecture (p. 1)
- AWS Cloud-Based Architecture (p. 1)
- How to Get Started (p. 2)

General Architecture

The following illustration shows a high-level view of the SPEKE content encryption architecture for on-premises products.

These are the main components of the preceding architecture:
- **Encryptor** – Provides the encryption technology. Receives encryption requests from its operator, and retrieves the required keys from the DRM key provider to secure the encrypted content.
- **DRM platform key provider** – Provides encryption keys to the encryptor through a SPEKE-compliant API. The provider also provides licenses to media players for decryption.
- **Player** – Requests keys from the same DRM platform key provider, which the player uses to unlock the content and serve it to its viewers.

AWS Cloud-Based Architecture

The following illustration shows the high-level architecture when SPEKE is used with services and features running in the AWS Cloud.
These are the main services and components:

- **Encryptor** – Provides the encryption technology in the AWS Cloud. The encryptor receives requests from its operator and retrieves the required encryption keys from the DRM key provider, through Amazon API Gateway, to secure the encrypted content. It delivers the encrypted content to an Amazon S3 bucket or through an Amazon CloudFront distribution.

- **AWS IAM and Amazon API Gateway** – Manages customer-trusted roles and proxy communication between the encryptor and the key provider. API Gateway provides logging capabilities and lets customers control their relationships with the encryptor and with the DRM platform. Customers enable key provider access through IAM role configuration. The API Gateway must reside in the same AWS Region as the encryptor.

- **AWS Certificate Manager** – (Optional) Provides certificate management for content key encryption. Encrypting content keys is the recommended practice for secure communication. The certificate manager must reside in the same AWS Region as the encryptor.

- **DRM platform key provider** – Provides encryption keys to the encryptor through a SPEKE-compliant API. The provider also provides licenses to media players for decryption.

- **Player** – Requests keys from the same DRM platform key provider, which the player uses to unlock the content and serve it to its viewers.

### How to Get Started

For additional introductory material about SPEKE, see *Are You New to SPEKE?* (p. 3).

**Are you a customer?**

Partner with an AWS Elemental DRM platform provider to get set up to use encryption. For details, see *Customer Onboarding* (p. 5).

**Are you a DRM platform provider or a customer with your own key provider?**

Expose a REST API for your key provider in compliance with the SPEKE specification. For details, see *SPEKE API Specification* (p. 8).
Are You New to SPEKE?

This section provides introductory material, for readers who are new to Secure Packager and Encoder Key Exchange (SPEKE).

For an introduction to SPEKE, watch the AWS SPEKE Webcast.

Related Services and Specifications

- **API Gateway Permissions** – How to control access to an API with AWS Identity and Access Management (AWS IAM) permissions.
- **AWS AssumeRole** – How to use AWS Security Token Service (AWS STS) to assume role functionality.
- **AWS Sigv4** – How to sign an HTTP request using Signature Version 4.
- **DASH-IF CPIX specification** – The DASH-IF Content Protection Information Exchange Format (CPIX) specification, which this SPEKE specification is based on.
- **DASH-IF System IDs** – The list of registered identifiers for DRM systems.
- **https://github.com/awslabs/speke-reference-server** – Example reference key provider to use with your AWS account, to help you get started with a SPEKE implementation in AWS.

Terminology

The following list defines the terminology used in this specification. Where possible, this specification follows the terminology used in the DASH-IF CPIX specification.

- **ARN** – Amazon Resource Name. Uniquely identifies an AWS resource.
- **Content key** – A cryptographic key used for encrypting part of the content.
- **Content provider** – A publisher who provides the rights and rules for delivering protected media. The content provider might also provide source media (mezzanine format, for transcoding), asset identifiers, key identifiers (KIDs), key values, encoding instructions, and content description metadata.
- **DRM** – Digital rights management. Used to protect copyrighted digital content from unapproved access.
- **DRM platform** – A system that provides DRM functionality and support to content encryptors and viewers, including providing DRM keys and licensing for content encryption and decryption.
- **DRM provider** – See DRM platform.
- **DRM system** – A standard for DRM implementations. Common DRM systems include Apple FairPlay, Google Widevine, and Microsoft PlayReady. DRM systems are used by content providers to secure digital content for delivery to viewers and for access by viewers. For a list of DRM systems that are registered with DASH-IF, see DASH-IF System IDs. The DASH-IF CPIX specification uses the term “DRM system” as defined here and, in some places, it uses “DRM system” to mean what this specification refers to as a DRM platform.
- **DRM solution** – See DRM platform.
- **DRM technology** – See DRM system.
- **Encryptor** – A media processing component that encrypts media content using keys obtained from the key provider. Encryptors typically also add DRM encryption signaling and metadata to the media. Encryptors are usually encoders, packagers, and transcoders.
• **Key provider** – The component of a DRM platform that exposes a SPEKE REST API to handle key requests. The key provider might be the key server itself, or it might be another component of the platform.

• **Key server** – The component of a DRM platform that maintains keys for content encryption and decryption.

• **Operator** – A person in charge of operating the overall system, including the encryptor and the key provider.

• **Player** – A media player operating on behalf of a viewer. Gets its information from different sources, including the media manifest files, media files, and DRM licenses. Requests licenses from the DRM platform on behalf of the viewers.
Customer Onboarding

Protect your content from unauthorized use by combining a Secure Packager and Encoder Key Exchange (SPEKE) digital rights management (DRM) key provider with your encryptor and with your media players. SPEKE defines the standard for communication between encryptors and packagers of media content and digital rights management (DRM) key providers. To onboard, you choose a DRM platform key provider and configure the communication between the key provider and your encryptors and players.

Topics
• Get On Board with a DRM Platform Provider (p. 5)
• SPEKE Support in AWS Services and Products (p. 5)

Get On Board with a DRM Platform Provider

The following Amazon partners provide third-party DRM platform implementations for SPEKE. For details about their offerings and information about how to contact them, follow the links to their Amazon Partner Network pages. Partners that don't have a link don't currently have an Amazon Partner Network page, but you can contact them directly. The partners can help you get set up to use their platforms.

• Axinom
• BuyDRM
• castLabs
• Conax AS
• EZDRM
• INKA Entworks
• Insys Video Technologies
• Intertrust Technologies
• Irdeto
• Kaltura
• NAGRA
• NEXTSCAPE, Inc.
• SeaChange
• Verimatrix
• Viaccess-Orca
• VUALTO
• WebStream

SPEKE Support in AWS Services and Products

This section lists the SPEKE support that is provided by AWS Media Services that run in the AWS Cloud and by AWS on-premises media products. These services and products are the encryptors in the SPEKE
content encryption architecture. Verify that your streaming protocol and the DRM system that you want are available for your service or product.

***AWS Elemental MediaConvert - Service That Runs in the AWS Cloud***

<table>
<thead>
<tr>
<th>Support matrix for protocol and DRM system</th>
<th>Microsoft PlayReady</th>
<th>Google Widevine</th>
<th>Apple FairPlay</th>
<th>AES-128</th>
<th>SAMPLE AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DASH</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apple HLS</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Microsoft Smooth</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMAF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

***AWS Elemental MediaPackage - Service That Runs in the AWS Cloud***

<table>
<thead>
<tr>
<th>Support matrix for protocol and DRM system</th>
<th>Microsoft PlayReady</th>
<th>Google Widevine</th>
<th>Apple FairPlay</th>
<th>AES-128</th>
</tr>
</thead>
<tbody>
<tr>
<td>DASH</td>
<td>✓ with key rotation</td>
<td>✓ with key rotation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apple HLS</td>
<td></td>
<td></td>
<td>✓ with key rotation</td>
<td>✓ with key rotation</td>
</tr>
<tr>
<td>Microsoft Smooth</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMAF Apple HLS</td>
<td></td>
<td></td>
<td>✓ with key rotation</td>
<td></td>
</tr>
</tbody>
</table>

***AWS Elemental Live - On-premises Product***

<table>
<thead>
<tr>
<th>Support matrix for protocol and DRM system</th>
<th>Microsoft PlayReady</th>
<th>Google Widevine</th>
<th>Apple FairPlay</th>
<th>AES-128</th>
</tr>
</thead>
<tbody>
<tr>
<td>DASH</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apple HLS TS</td>
<td></td>
<td></td>
<td>✓ with key rotation</td>
<td>✓ with key rotation</td>
</tr>
<tr>
<td>Apple HLS fMP4</td>
<td></td>
<td></td>
<td>✓ with key rotation</td>
<td></td>
</tr>
</tbody>
</table>

***AWS Elemental Delta - On-premises Product***

<table>
<thead>
<tr>
<th>Support matrix for protocol and DRM system</th>
<th>Microsoft PlayReady</th>
<th>Google Widevine</th>
<th>Apple FairPlay</th>
<th>AES-128</th>
</tr>
</thead>
<tbody>
<tr>
<td>DASH</td>
<td>✓ with key rotation</td>
<td>✓ with key rotation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apple HLS</td>
<td></td>
<td></td>
<td>✓ with key rotation</td>
<td>✓ with key rotation</td>
</tr>
<tr>
<td>Microsoft Smooth</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMAF Apple HLS</td>
<td></td>
<td></td>
<td>✓ with key rotation</td>
<td></td>
</tr>
<tr>
<td>Support matrix for protocol and DRM system</td>
<td>Microsoft PlayReady</td>
<td>Google Widevine</td>
<td>Apple FairPlay</td>
<td>AES-128</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>---------</td>
</tr>
<tr>
<td>DASH</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apple HLS</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Microsoft Smooth</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMAF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
SPEKE API Specification

This is the REST API specification for Secure Packager and Encoder Key Exchange (SPEKE). Use this specification to provide DRM copyright protection for customers who use encryption.

In a video streaming workflow, the encryption engine communicates with the DRM platform key provider to request content keys. These keys are highly sensitive, so it is critical that the key provider and encryption engine establish a highly secure, trusted communication channel. You can also encrypt the content keys in the document for more secure, end-to-end encryption.

This specification addresses the following goals:

- Define a simple, trusted, highly secure interface that DRM vendors and customers can use to integrate with encryptors when content encryption is required.
- Cover VOD and live workflows, and include the error conditions and the authentication mechanisms that are required for robust, highly secure communication between encryptors and DRM key provider endpoints.
- Include support for HLS, MSS, and DASH packaging and their common DRM systems: FairPlay, PlayReady, and Widevine/CENC.
- Keep the specification simple and extensible, to support future DRM systems.
- Use a simple REST API.

Note
The code examples in this specification are for illustration purposes only. You can't run the examples because they aren't part of a complete SPEKE implementation.

Topics
- Authentication (p. 8)
- SPEKE API (p. 9)

Authentication

SPEKE requires authentication for on-premises products and for services and features that run in the AWS Cloud.

Topics
- Authentication for AWS Cloud Implementations (p. 8)
- Authentication for On-premises Products (p. 9)

Authentication for AWS Cloud Implementations

SPEKE requires AWS authentication through IAM roles for use with an encryptor. IAM roles are created by the DRM provider or by the operator who owns the DRM endpoint in an AWS account. Each role is assigned an Amazon Resource Name (ARN), which the AWS Elemental service operator provides on the service console when requesting encryption. The role's policy permissions must be configured to give permission to access the key provider API and no other AWS resource access. When the encryptor contacts the DRM key provider, it uses the role ARN to assume the role of the key provider account holder, which returns temporary credentials for the encryptor to use to access the key provider.
One common implementation is for the operator or DRM platform vendor to use Amazon API Gateway in front of the key provider, and then enable AWS Identity and Access Management (AWS IAM) authorization on the API Gateway resource. You can use the following policy definition example and attach it to a new role to give permissions to the appropriate resource. In this case, the permissions are for all API Gateway resources:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:us-west-2:*:*/*/GET/*"
            ]
        }
    ]
}
```

Finally, the role requires the addition of a trust relationship, and the operator must be able to select the service.

The following example shows a role ARN that is created for accessing the DRM key provider:

```
arn:aws:iam::2949266363526:role/DRMKeyServer
```

For more information about the creation of a role, see AWS AssumeRole. For more information about signing a request, see AWS Sigv4.

**Authentication for On-premises Products**

For on-premises products, we recommend that you use SSL/TLS and digest authentication for the best security, but at a minimum you should use basic authentication over HTTPS.

Both types of authentication use the Authorization header in the HTTP request:

- **Digest authentication** – The authorization header consists of the identifier Digest followed by a series of values that authenticate the request. Specifically, a response value is generated through a series of MD5 hash functions that include a unique, one-time-use nonce from the server that is used to ensure that the password travels securely.

- **Basic authentication** – The authorization header consists of the identifier Basic followed by a base-64 encoded string that represents the user name and password, separated by a colon.

For information about basic and digest authentication, including detailed information about the header, see the Internet Engineering Task Force (IETF) specification RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication.

**SPEKE API**

To be SPEKE-compliant, your DRM key provider must expose the REST API described in this specification. The encryptor makes API calls to your key provider.
Note
The code examples in this specification are for illustration purposes only. You can’t run the examples because they aren’t part of a complete SPEKE implementation.

Secure Packager and Encoder Key Exchange uses the DASH Industry Forum Content Protection Information Exchange Format (DASH-IF-CPIX) data structure definition for key exchange, with some restrictions. DASH-IF-CPIX defines a schema to provide an extensible, multi-DRM exchange from the DRM platform to the encryptor. This enables content encryption for all adaptive bitrate packaging formats at the time of content compression and packaging. Adaptive bitrate packaging formats include HLS, DASH, and MSS.

For detailed information about the exchange format, see the DASH Industry Forum CPIX specification at https://dashif.org/docs/DASH-IF-CPIX-v2-0.pdf.

Topics
• Customizations and Constraints to the DASH-IF Specification (p. 10)
• Standard Payload Components (p. 11)
• Live Workflow Method Call Examples (p. 12)
• VOD Workflow Method Call Examples (p. 16)
• Content Key Encryption (p. 18)
• Heartbeat (p. 21)
• Overriding the Key Identifier (KID) (p. 21)

Customizations and Constraints to the DASH-IF Specification

The DASH-IF CPIX specification, https://dashif.org/docs/DASH-IF-CPIX-v2-0.pdf, supports a number of use cases and topologies. The SPEKE API specification adheres to the CPIX specification with the following customizations and constraints:

• SPEKE follows the Encryptor Consumer workflow.
• For encrypted content keys, SPEKE applies the following restrictions:
  • SPEKE doesn’t support digital signature verification (XMLDSIG) for request or response payloads.
  • SPEKE requires 2048 RSA-based certificates.
• For rotating key workflows, SPEKE requires the ContentKeyUsageRule filter, KeyPeriodFilter. SPEKE ignores all other ContentKeyUsageRule settings.
• SPEKE omits the UpdateHistoryItemList functionality. If the list is present in the response, SPEKE ignores it.
• SPEKE supports key rotation. SPEKE uses only the ContentKeyPeriod @index to track the key period.
• To support MSS PlayReady, SPEKE uses a custom parameter under the DRMSystem tag, SPEKE:ProtectionHeader.
• For HLS packaging, if the URIExtXKey is present in the response, then it must contain the full data to add in the URI parameter of the EXT-X-KEY tag of an HLS playlist, with no further signaling requirement.
• For HLS playlist, under the DRMSystem tag, SPEKE provides the optional custom parameters speke:KeyFormat and speke:KeyFormatVersions, for the values of the KEYFORMAT and KEYFORMATVERSIONS parameters of the EXT-X-KEY tag.

The HLS initialization vector (IV) always follows segment number unless explicitly specified by the operator.
• When requesting keys, the encryptor might use the optional `@explicitIV` attribute on the `ContentKey` element. The key provider can respond with an IV using `@explicitIV`, even if the attribute is not included in the request.

• The encryptor creates the key identifier (KID), which stays the same for any given content ID and key period. The key provider includes the KID in its response to the request document.

• The key provider might include a value for the `Speke-User-Agent` response header, to identify itself for debugging purposes.

• SPEKE does not currently support multiple tracks or keys per content.

The SPEKE-compliant encryptor acts as a client and sends POST operations to the key provider endpoint. The encryptor might send a periodic heartbeat request to ensure that the connection between the encryptor and the key provider endpoint is healthy.

**Standard Payload Components**

In any SPEKE request, the encryptor can request responses for one or more DRM systems. The encryptor specifies the DRM systems in `<cpix:DRMSystemList>` of the request payload. Each system specification includes the key and indicates the type of response to return.

The following example shows a DRM system list with a single DRM system specification:

```xml
<!-- HLS AES-128 (systemId is implementation specific) -->
<cpix:DRMSystemList>
  <cpix:DRMSystem kid="98ee5596-cd3e-a20d-163a-e382420c6eff"
    systemId="81376844-f976-481e-a84e-cc25d39b0b33">
    <cpix:UriExtXKey/>
    <speke:KeyFormat/>
    <speke:KeyFormatVersions/>
  </cpix:DRMSystem>
</cpix:DRMSystemList>
```

The following table lists the main components of each `<cpix:DRMSystem>`.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>systemId or schemeId</td>
<td>Unique identifier for the DRM system type, as registered with the DASH IF organization. For a list, see DASH-IF System IDs.</td>
</tr>
<tr>
<td>kid</td>
<td>The key ID. This is not the actual key, but an identifier that points to the key in a hash table.</td>
</tr>
<tr>
<td><code>&lt;cpix:UriExtXKey&gt;</code></td>
<td>Requests a standard unencrypted key. The key response type must be either this or the PSSH response.</td>
</tr>
<tr>
<td><code>&lt;cpix:PSSH&gt;</code></td>
<td>Requests a Protection System Specific Header (PSSH). This type of header contains a reference to the kid, the systemID, plus custom data for the DRM vendor, as part of Common Encryption (CENC). The key response type must be either this or the UriExtXKey response.</td>
</tr>
</tbody>
</table>

*Example Requests for Standard Key and for PSSH*
The following example shows part of a sample request from the encryptor to the DRM key provider, with the main components highlighted. The first request is for a standard key, while the second request is for a PSSH response:

Example Responses for Standard Key and for PSSH

The following example shows the corresponding response from the DRM key provider to the encryptor:

Request Syntax Example
The following URL is an example and does not indicate a fixed format:

```
POST https://speke-compatible-server/speke/v1.0/copyProtection
```

**Request Body**

A CPIX element.

**Request Headers**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Occurs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Authorization</td>
<td>String</td>
<td>1..1</td>
<td>See AWS Sigv4</td>
</tr>
<tr>
<td>X-Amz-Security-Token</td>
<td>String</td>
<td>1..1</td>
<td>See AWS Sigv4</td>
</tr>
<tr>
<td>X-Amz-Date</td>
<td>String</td>
<td>1..1</td>
<td>See AWS Sigv4</td>
</tr>
<tr>
<td>Content-Type</td>
<td>String</td>
<td>1..1</td>
<td>application/xml</td>
</tr>
</tbody>
</table>

**Response Headers**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Occurs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speke-User-Agent</td>
<td>String</td>
<td>1..1</td>
<td>String that identifies the key provider</td>
</tr>
<tr>
<td>Content-Type</td>
<td>String</td>
<td>1..1</td>
<td>application/xml</td>
</tr>
</tbody>
</table>

**Request Response**

<table>
<thead>
<tr>
<th>HTTP CODE</th>
<th>Payload Name</th>
<th>Occurs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (Success)</td>
<td>CPIX</td>
<td>1..1</td>
<td>DASH-CPIX payload response</td>
</tr>
<tr>
<td>4XX (Client error)</td>
<td>Client error message</td>
<td>1..1</td>
<td>Description of the client error</td>
</tr>
<tr>
<td>5XX (Server error)</td>
<td>Server error message</td>
<td>1..1</td>
<td>Description of the server error</td>
</tr>
</tbody>
</table>

**Note**

The examples in this section do not include content key encryption. For information about how to add content key encryption, see the section called “Content Key Encryption” (p. 18).

**Live Example Request Payload with Keys in the Clear**

The following example shows a typical live request payload from the encryptor to the DRM key provider:
Secure Packager and Encoder Key Exchange
API Specification Partner and Customer Guide
Live Workflow Method Call Examples

The following example shows a typical response payload from the DRM key provider:

Live Example Response Payload with Keys in the Clear
VOD Workflow Method Call Examples

Request Syntax Example

The following URL is an example and does not indicate a fixed format.

```
POST https://speke-compatible-server/speke/v1.0/copyProtection
```

Request Body

A CPIX element.

Response Headers

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Occurs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speke-User-Agent</td>
<td>String</td>
<td>1..1</td>
<td>String that identifies the key provider</td>
</tr>
<tr>
<td>Content-Type</td>
<td>String</td>
<td>1..1</td>
<td>application/xml</td>
</tr>
</tbody>
</table>

Request Response

<table>
<thead>
<tr>
<th>HTTP CODE</th>
<th>Payload Name</th>
<th>Occurs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (Success)</td>
<td>CPIX</td>
<td>1..1</td>
<td>DASH-CPIX payload response</td>
</tr>
<tr>
<td>4XX (Client error)</td>
<td>Client error message</td>
<td>1..1</td>
<td>Description of the client error</td>
</tr>
<tr>
<td>5XX (Server error)</td>
<td>Server error message</td>
<td>1..1</td>
<td>Description of the server error</td>
</tr>
</tbody>
</table>

Note

The examples in this section do not include content key encryption. For information on how to add content key encryption, see the section called “Content Key Encryption” (p. 18).

VOD Example Request Payload with Keys in the Clear

The following example shows a basic VOD request payload from the encryptor to the DRM key provider:
Secure Packager and Encoder Key Exchange
API Specification Partner and Customer Guide
VOD Workflow Method Call Examples

VOD Example Response Payload with Keys in the Clear

The following example shows a basic VOD response payload from the DRM key provider:
Content Key Encryption

You can optionally add content key encryption to your SPEKE implementation. Content key encryption guarantees full end-to-end protection by encrypting the content keys for transit, in addition to...
encrypting the content itself. If you don’t implement this for your key provider, you rely on the transport layer encryption plus strong authentication for security.

To use content key encryption for encryptors running in AWS Cloud, customers import certificates into the AWS Certificate Manager and then use the resulting certificate ARNs for their encryption activities. The encryptor uses the certificate ARNs and the ACM service to provide encrypted content keys to the DRM key provider.

**Restrictions**

SPEKE supports content key encryption as specified in the DASH-IF CPIX specification with the following restrictions:

- SPEKE doesn’t support digital signature verification (XMLDSIG) for request or response payloads.
- SPEKE requires 2048 RSA-based certificates.

These restrictions are also listed in the section called “Customizations and Constraints” (p. 10).

**Implement content key encryption**

To provide content key encryption, include the following in your DRM key provider implementations:

- Handle the element `<cpix:DeliveryDataList>` in the request and response payloads.
- Provide encrypted values in the `<cpix:ContentKeyList>` of the response payloads.

For more information about these elements, see the DASH-IF CPIX 2.0 specification.

**Example Content Key Encryption Element `<cpix:DeliveryDataList>` in the Request Payload**

The following example highlights the added `<cpix:DeliveryDataList>` element in bold:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX id="example-test-doc-encryption"
   xmlns:cpix="urn:dashif:org:cpix"
   xmlns:speke="urn:aws:amazon:com:speke">
   <cpix:DeliveryDataList>
     <cpix:DeliveryData id="<ORIGIN SERVER ID>">
       <cpix:DeliveryKey>
         <ds:X509Data>
         </ds:X509Data>
       </cpix:DeliveryKey>
     </cpix:DeliveryData>
   </cpix:DeliveryDataList>
   <cpix:ContentKeyList>...
   </cpix:ContentKeyList>
</cpix:CPIX>
```

**Example Content Key Encryption Element `<cpix:DeliveryDataList>` in the Response Payload**

The following example highlights the added `<cpix:DeliveryDataList>` element in bold:

```xml
<cpix:CPIX xmlns:cpix="urn:dashif:org:cpix"
   xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
   xmlns:speke="urn:aws:amazon:com:speke" id="hls_test_001">
   <cpix:DeliveryDataList>
     ...
   </cpix:DeliveryDataList>
</cpix:CPIX>
```
Example Content Key Encryption Element `<cpix:ContentKeyList>` in the Response Payload

The following example shows encrypted content key handling in the `<cpix:ContentKeyList>` element of the response payload. This uses the `<pskc:EncryptedValue>` element:

```
<cpix:DeliveryDataList>
    <cpix:DeliveryData id="<ORIGIN SERVER ID>">
        <cpix:DeliveryKey>
            <ds:X509Data>
                <ds:X509Certificate>
                    <ds:X509Certificate>
                </ds:X509Certificate>
            </ds:X509Data>
        </cpix:DeliveryKey>
        <cpix:DocumentKey Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:EncryptedValue>
                        <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p" />
                        <enc:CipherData>
                            <enc:CipherValue>NJYebfvJ2TdMm3k6v+rlNvYb0N0TJoTLBBdgbpe8nmiELefp82Sk7MkqTn21mQBPB</enc:CipherValue>
                        </enc:CipherData>
                    </pskc:EncryptedValue>
                    <pskc:ValueMAC>t9lW4WCebfS1GP+dh0IicMs+2+jnrAmfDa4WU6VGHC4=</pskc:ValueMAC>
                </pskc:Secret>
            </cpix:Data>
        </cpix:DocumentKey>
    </cpix:DeliveryData>
</cpix:DeliveryDataList>
```

```
Example Content Key Encryption Element `<cpix:ContentKeyList>` in the Response Payload

The following example shows encrypted content key handling in the `<cpix:ContentKeyList>` element of the response payload. This uses the `<pskc:EncryptedValue>` element:

```
<cpix:DeliveryDataList>
    <cpix:DeliveryData id="<ORIGIN SERVER ID>">
        <cpix:DeliveryKey>
            <ds:X509Data>
                <ds:X509Certificate>
                    <ds:X509Certificate>
                </ds:X509Certificate>
            </ds:X509Data>
        </cpix:DeliveryKey>
        <cpix:DocumentKey Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:EncryptedValue>
                        <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p" />
                        <enc:CipherData>
                            <enc:CipherValue>NJYebfvJ2TdMm3k6v+rlNvYb0N0TJoTLBBdgbpe8nmiELefp82Sk7MkqTn21mQBPB</enc:CipherValue>
                        </enc:CipherData>
                    </pskc:EncryptedValue>
                    <pskc:ValueMAC>T9lw4WcBef81Gp+dh0IicMs+2+jnrAmfDa4WU6VGHC4=</pskc:ValueMAC>
                </pskc:Secret>
            </cpix:Data>
        </cpix:DocumentKey>
    </cpix:DeliveryData>
</cpix:DeliveryDataList>
```
Secure Packager and Encoder Key Exchange
API Specification Partner and Customer Guide
Heartbeat

By comparison, the following example shows a similar response payload with the content key delivered unencrypted, as a clear key. This uses the `<pskc:PlainValue>` element:

```xml
<cpix:ContentKeyList>
  <cpix:ContentKey explicitIV="OFj2IjCsPfjFmAXmQxLGPw==" kid="682681c8-69fa-4434-9f9f-1a7f5389ec02">
    <cpix:Data>
      <pskc:Secret>
        <pskc:PlainValue>5dGAgwGuUYu4dHeHtNlxJw==</pskc:PlainValue>
      </pskc:Secret>
    </cpix:Data>
  </cpix:ContentKey>
</cpix:ContentKeyList>
```

Heartbeat

Request Syntax Example

The following URL is an example and does not indicate a fixed format:

```
GET https://speke-compatible-server/speke/v1.0/heartbeat
```

Request Response

<table>
<thead>
<tr>
<th>HTTP CODE</th>
<th>Payload Name</th>
<th>Occurs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (Success)</td>
<td>statusMessage</td>
<td>1..1</td>
<td>Message that describes the status</td>
</tr>
</tbody>
</table>

Overriding the Key Identifier (KID)

The encryptor creates a new key identifier (KID) each time that it rotates keys. It passes the KID to the DRM key provider in its requests. Almost always, the key provider responds using the same KID, but it can provide a different value for the KID in the response.

The following is an example request with the KID `11111111-1111-1111-1111-111111111111`:

```xml
  <cpix:ContentKeyList>
    <cpix:ContentKey kid="11111111-1111-1111-1111-111111111111" />
  </cpix:ContentKeyList>
  <cpix:DRMSYSTEMList>
    <!-- Common encryption (Widevine)-->-
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="edef8ba9-79d6-4ace-a3c8-27dcd51d21ed" />
    <cpix:PSSH />
  </cpix:DRMSystemList>
</cpix:CPIX>
```
The following response overrides the KID to 22222222-2222-2222-2222-222222222222:

```xml
    <cpix:ContentKeyList>
        <cpix:ContentKey explicitIV="ASgwx9pQ2/2lnDzJsUxWcQ==" kid="22222222-2222-2222-2222-222222222222">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>p3dWaHARtL97MpT7TE916w==</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
    </cpix:ContentKeyList>
    <cpix:DRMSystemList>
        <cpix:DRMSystem kid="22222222-2222-2222-2222-222222222222" systemId="edef8ba9-79d6-4ace-a3c8-27dcd51d21ed">
            <cpix:PSSH>AAAAanBzc2gAAAAA7e+LqXnWSs6jyCfc1R0h7QAAAEOIARIQesIcblaNbb7Dji6sAtKZsRoNd2lkZXpbmVfdGVzdCIfa2VSLWlkOmVTSWNibGFOYmI3RGppNyNcpix:PSSH></cpix:PSSH>
        </cpix:DRMSystem>
    </cpix:DRMSystemList>
    <cpix:ContentKeyPeriodList>
        <cpix:ContentKeyPeriod id="keyPeriod_0909829f-40ff-4625-90fa-75da3e53278f" index="1" />
    </cpix:ContentKeyPeriodList>
</cpix:CPIX>
```
# Document History

The following table describes the changes to the SPEKE documentation.

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support matrix - AWS Elemental Delta (p. 5)</td>
<td>Added an AWS Elemental Delta support matrix.</td>
<td>February 7, 2019</td>
</tr>
<tr>
<td>Updates to DRM platform providers (p. 5)</td>
<td>Added links and new partner information to the DRM platform provider list.</td>
<td>January 24, 2019</td>
</tr>
<tr>
<td>Include third-party encryptors (p. 1)</td>
<td>Updated the architecture and descriptions to account for third-party encryptors.</td>
<td>November 20, 2018</td>
</tr>
<tr>
<td>Content Key Encryption (p. 18)</td>
<td>Added the option to encrypt content keys. Prior to this, Secure Packager and Encoder Key Exchange supported clear key delivery only.</td>
<td>October 30, 2018</td>
</tr>
<tr>
<td>Support matrix - AWS Elemental Live (p. 5)</td>
<td>Added an AWS Elemental Live support matrix.</td>
<td>September 27, 2018</td>
</tr>
<tr>
<td>Standard payload components (p. 11)</td>
<td>Added a section that defines the main elements in the JSON payload.</td>
<td>September 27, 2018</td>
</tr>
<tr>
<td>KID override (p. 21)</td>
<td>Added a section about KID overrides by a key provider.</td>
<td>September 27, 2018</td>
</tr>
<tr>
<td>Corrected links to DASH-IF site (p. 3)</td>
<td>Corrected links to the DASH IF site for the CPIX specification and for the system IDs page.</td>
<td>September 27, 2018</td>
</tr>
<tr>
<td>Release copy for AWS Elemental Live (p. 1)</td>
<td>Updated the SPEKE documentation to include AWS Elemental products.</td>
<td>July 20, 2018</td>
</tr>
<tr>
<td>CMAF (p. 5)</td>
<td>Updated the support matrix tables for services to include the Common Media Application Format (CMAF).</td>
<td>June 27, 2018</td>
</tr>
<tr>
<td>First release (p. 1)</td>
<td>First release of Secure Packager and Encoder Key Exchange (SPEKE), a specification for communication between a content encryptor and a DRM key provider. The DRM key provider exposes a Secure Packager and Encoder Key Exchange API to handle incoming key requests.</td>
<td>November 27, 2017</td>
</tr>
</tbody>
</table>
AWS glossary

For the latest AWS terminology, see the AWS glossary in the AWS General Reference.